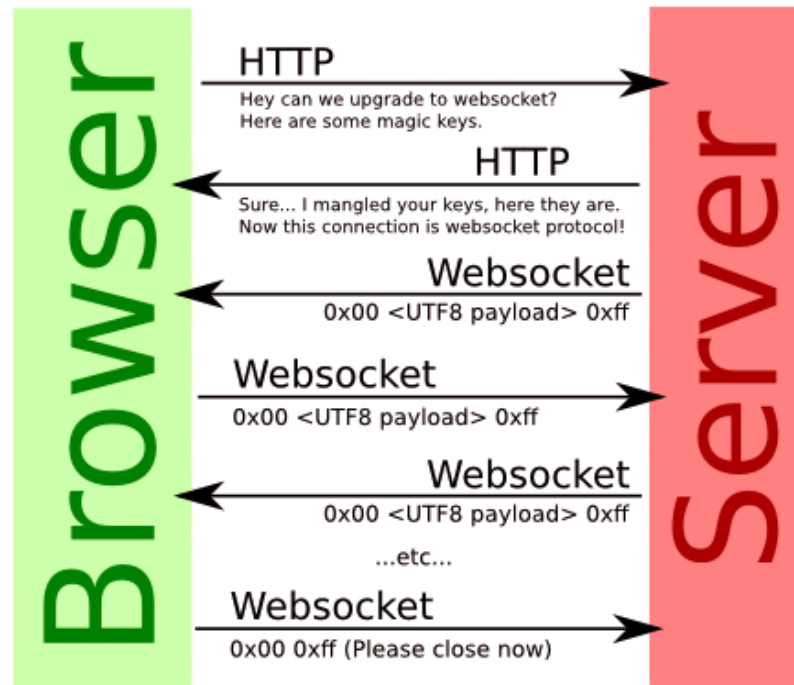


Protocolos web

HTTP & WebSocket



Version PDF des slides

HTTP ?

- **HyperText Transfert Protocole**
- **Objectif:** accéder à et manipuler des ressources (via des URL)
- Fonctionne par dessus TCP/IP
- Basé lignes de **texte**
 - On peut communiquer avec un serveur HTTP via `telnet` sur le port 80
- **Sans état**
 - N'a aucune mémoire des transactions passées
- Fonctionnement en mode client / serveur
 - **Requête / réponse**
- Existe en version sécurisée (HTTPS)

Historique

- **1990:** HTTP/0.9
 - **Seulement la méthode GET** (récupération d'une ressource)
 - Pas de code de retour ou de type de fichiers : ne sait renvoyer que du `text/plain`
- **1996:** HTTP/1.0 ([RFC 1945](#))
 - **Nouvelles méthodes** de communication (POST, PUT, etc.)
 - Entêtes, types MIME, authentification, etc.
- **1997:** HTTP/1.1 ([RFC 2068](#))
 - Modernisation: **connexions persistantes**, meilleur gestion du cache, requêtes partielles, compression ,etc.

Historique

- **1999:** Maj de HTTP/1.1 ([RFC 2616](#))
 - **Négociation de contenu**
- **2011:** WebSocket ([RFC 6455](#))
 - **Communication full-duplex**
 - Envoi de données en mode "Push"
 - Passage des serveurs proxy facilité
- **2014:** Clarification de HTTP/1.1 ([RFCs 7230-7237](#))
- **2015:** HTTP/2 ([RFC 7540](#))
 - Objectif principal: **accélérer le web** (dérivé de google SPDY) via le multiplexage des requêtes
- **2022:** HTTP/3 ([RFC 9114](#))
 - Abandon de TCP en faveur de [QUIC](#) (basé sur UDP) pour réduire la latence

Pourquoi étudier HTTP ?

- C'est la **base de la communication** entre le navigateur et le serveur
- **Nécessaire** coté client lors par exemple de l'envoi de requêtes AJAX
- **Indispensable** coté serveur pour
 - Récupérer les données envoyées par le client
 - Gérer les sessions
 - Gérer la mise en cache des informations
 - etc.

Exemple de requête HTTP

```
GET / HTTP/1.1
Host: www.perdu.com
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: fr;q=0.8,en;q=0.6
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/41.0.2272.101 Safari/537.36
```

Attention: la requête termine par une ligne vide

Exemple de réponse HTTP

```
HTTP/1.1 200 OK
Date: Thu, 26 Mar 2015 14:48:50 GMT
Server: Apache
Last-Modified: Tue, 02 Mar 2010 18:52:21 GMT
Etag: "cc-480d5dd98a340"
Content-Type: text/html
Content-Length: 204
Accept-Ranges: bytes

<html><head><title>Vous Etes Perdu ?</title></head><body><h1>Perdu sur l'Internet ?</h1>
<h2>Pas de panique, on va vous aider</h2>
<strong><pre>    * <----- vous êtes ici</pre></strong></body></html>
```

Attention: une ligne vide sépare les entêtes des données renvoyées par le serveur

Structure d'une requête

1. Ligne de requête: `METHOD URL [HTTP-version]` *

- Ex: `GET /index.html HTTP/1.1`

2. Entêtes (ordre conseillé mais pas obligatoire)

- Entêtes **généraux**: applicables aux requêtes **et** réponses
 - Ex: `Date: Thu, 26 Mar 2015 14:48:50 GMT`)
- Entêtes de **requête**: spécifiques aux requêtes
 - Ex: `Accept: text/html`
- Entêtes d'**entité**: méta-informations concernant le corps du message (pour les requêtes **et** les réponses)
 - Ex: `Content-Length: 204`

3. Ligne vide

4. Corps du message (**données**)

- Ex: texte, code html, données de formulaire, image, etc.

* Par défaut c'est la version 1.0 de HTTP qui est utilisée

Les principales méthodes HTTP

- **OPTIONS** : demande les **options de communication** disponibles
- **GET** : demande des **informations** (entêtes) et des **données** (corps de la réponse) concernant la ressource située à l'URL spécifiée
 - Utilisé lorsque vous consultez une page web
- **HEAD** : demande des **informations** (entêtes) concernant la ressource située à l'URL spécifiée
- **POST** : **envoie des données** (contenu d'un formulaire) qui seront traitées par le script / programme situé à l'URL
 - Utilisé lorsque vous envoyez les données d'un formulaire (généralement)
- **PUT** : **stocke** des données à l'URL spécifiée
 - Utilisé lorsque vous envoyez un fichier vers un serveur
- **PATCH** : applique des **modifications partielles** à une ressource
- **DELETE** : **supprime** les données situées l'URL spécifiée
- **TRACE** : demande au serveur de retourner au client les données qui lui ont été envoyées (**écho**)

Rappel sur les URL

- **Absolute** : schéma://utilisateur:motdepasse@domaine:port/chemin?requête#fragment
 - Ex: `http://joe:bar@www.univ-nantes.fr:80/polytech/dpts/info?enseignant=perreiradasilva-m&cours=technos-web#slide4`
- **Relative** : chemin?requête#fragment
 - Ex1: `ici`
 - Ex2: `/ici`
 - Ex3: `./ici?query=something`
 - Ex4: `../la/fichier.html`
- Ne sont codées qu'à partir d'un **jeu limité** de caractères. Pour le reste, on utilise le **percent encoding**

Encodage des caractères

- Représentation numérique des caractères
 - Ex: 'A' = 0x41 en ASCII (hexadécimal)
- **ASCII**: caractères de base (US) codés sur 7bits
- **Variantes de l'ASCII** sur 8 bits
 - Nécessité de préciser le **jeu de caractères** (character set / codepage)
 - Ex: ISO **8859-1** (europe occidentale) et **8859-15** (variation avec €)
- **UTF-8**: encodage universel à taille variable (**RFC 3629**)
 - De 8 à 32 bits, rétrocompatible avec ASCII
 - Ex: 'A' = 0x41 , 'à' = 0xC3 0xA0 , '€' = 0xE2 0x82 0xAC

Percent-encoding

- Les URI ne sont codées qu'à partir d'un jeu limité de caractères ASCII
 - Caractères réservés : ont une signification particulière

! # \$ % & ' () * + , / : ; = ? @ []

- Caractères non réservés : pas de signification particulière

A B C D E F G H I J K L M N O P Q R S T
U V W X Y Z
a b c d e f g h i j k l m n o p q r s t
u v w x y z
0 1 2 3 4 5 6 7 8 9 - _ . ~

Percent-encoding (suite)

- Pour tous les autres caractères ou pour utiliser les caractères réservés sans qu'ils ne soient interprétés on utilise le percent encoding
 - Règle générale: %code_UTF-8 (en hexadécimal)
 - Ex1: espace = %20
 - Ex2: % = %25
 - Ex3: € = %E2%82%AC
 - Même chose pour les caractères réservés

!	#	\$	&	'	()	*	+
%21	%23	%24	%26	%27	%28	%29	%2A	%2B

,	/	:	;	=	?	@	[]
%2C	%2F	%3A	%3B	%3C	%3F	%40	%5B	%5D

Les types MIME (Content-type)

- Multipurpose Internet Mail Extensions (rfc 2045-2046)
 - Extension du format des emails pour supporter autre chose que le texte ASCII
 - Définit le format du contenu du mail et de ses pièces jointes
- Type MIME : partie Content-type / Accept des entêtes HTTP
 - Syntaxe : type/sous-type
 - Exemples :
 - Texte: text/plain , text/html
 - Fichiers pluri-usages: application/pdf , application/xml
 - Multimédia: audio/mpeg , image/jpeg , video/mp4
 - Etc.

Les entêtes généraux

- `Cache-Control` : permet de spécifier une politique de cache que **doivent** respecter clients ou serveurs
 - Ex: `Cache-control: no-cache`
- `Connection` : contrôle de la connexion HTTP (fermer, garder ouverte, etc.)
 - Ex: `Connection: close`
- `Date` : date d'envoi du message
 - Ex: `Date: Thu, 26 Mar 2015 14:48:50 GMT`
- `Pragma` : directive spécifique, plus utilisée
- `Trailer` : déclare qu'un certain nombre de champs seront présents à la fin du message plutôt que dans le header (transfert par blocs seulement)
 - Ex: `Trailer: Max-Forwards`

Les entêtes généraux

- `Transfer-Encoding` : méthode d'encodage des données
 - Valeurs autorisées: `chunked`, `compress`, `deflate`, `gzip` ou `identity`
- `Upgrade` : demande un changement de protocole
 - Ex: `Upgrade: websocket`
- `Via` : utilisé par les passerelles et proxy pour spécifier des protocoles intermédiaires
- `Warning` : informations complémentaires concernant le message
 - Ex: `Warning: 112 Disconnected Operation`

Requête : les entêtes spécifiques

- **Accept** : types MIME acceptés
 - Ex: `Accept: text/plain; q=0.5, text/html; q=0.8`
- **Accept-Charset** : encodages de caractères acceptés
 - Ex: `Accept-Charset: iso-8859-5, unicode-1-1;q=0.8`
- **Accept-Encoding** : types de compression (éventuelle) acceptés
 - Ex: `Accept-Encoding: gzip;q=1.0, identity; q=0.5, *;q=0`
- **Accept-Language** : langues acceptées
 - Ex: `Accept-Language: fr; q=1.0, en; q=0.5`

Pour tous les entêtes de type **Accept** , on peut spécifier une préférence **q** ($q \in [0,1]$) qui sera utile lors de la négociation de contenu.

- **Authorization** : informations d'authentification auprès d'un serveur web
 - Ex: `Authorization: Basic QWxhZGRpbjpvvcGVuIHNlc2FtZQ==`
- **Expect** : demande d'un comportement particulier du serveur
 - Ex: `Expect : 100-continue`

Requête : les entêtes spécifiques

- **From** : adresse mail de l'utilisateur du client
 - Ex: `From: matthieu.perreiradasilva@univ-nantes.fr`
- **Host** : adresse (et port) de l'hôte à qui est destiné la requête
 - Ex: `Host: web.polytech.univ-nantes.fr:80`
- **If-Match** : rend la requête conditionnelle à un tag d'entité **ETag**
 - Ex: `If-Match : "686897696a7c876b7e"`
- **If-Modified-Since** : rend la requête conditionnelle à une date de modification
 - Ex: `If-Modified-Since: Fri, 03 Apr 2015 12:00:00 GMT+2`
- **If-None-Match** : inverse de **If-Match**
- **If-Range** : requête conditionnelle (date ou Etag) sur une partie d'une ressource (cf. header **Range**)
- **If-Unmodified-Since** : Inverse de **If-Modified-Since**

Requête : les entêtes spécifiques

- **Max-Forward** : nombre maximal de fois qu'un message peut être transféré (cf. méthode **TRACE**)
 - Ex: **Max-Forwards** : 5
- **Proxy-Authorization** : informations d'authentification auprès d'un proxy
 - Ex: **Proxy-Authorization**: Basic dXNlbWU6dGVzdA==
- **Range** : ne demande qu'une partie d'une ressource
 - Ex: **Range**: bytes=500-999
- **Referer** : adresse à partir de laquelle la requête provient
 - Ex: **Referer**: http://www.univ-nantes.fr/index.html
- **TE** : indique au serveur quel type de transfert de données par bloc le client supporte
 - Ex: **TE**: deflate
- **User-Agent** : informations identifiant l'agent utilisateur (navigateur) utilisé.
 - Ex: Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.2; Win64; x64; Trident/6.0)

Les entêtes d'entité

- `Allow` : liste des méthode supportée pour accéder à la ressource
 - Ex: `Allow: GET, HEAD, PUT`
- `Content-Encoding` : type d'encodage utilisée pour encoder l'entité (ressource)
 - Ex: `Content-Encoding: gzip`
- `Content-Language` : langue utilisée pour représenter l'entité
 - Ex: `Content-Language: fr`
- `Content-Length` : taille de la ressource (en octets)
 - Ex: `Content-Length: 6894`
- `Content-Location` : URL à partir de laquelle on peut accéder à la ressource (si différente de l'URL de la requête)
 - Ex: `Content-Location: http://www.univ-nantes.fr/content.json`

Les entêtes d'entité

- **Content-MD5** : hash MD5 de la ressource, pour vérification éventuelle de son intégrité
 - Ex: **Content-MD5** : 8c2d46911f3f5a326455f0ed7a8ed3b3
- **Content-Range** : En cas d'envoi partiel de la ressource, indique la plage de données envoyée.
 - Ex: **Content-Range** : bytes 500-999/1234
- **Content-Type** : Type MIME de la ressource envoyée, suivi optionnellement de paramètres
 - Ex: **Content-Type**: text/html; charset=utf-8
- **Expires** : indique la date/heure à partir de laquelle la réponse est considérée comme obsolète
 - Ex: **Expires**: Wed, 01 Apr 2015 23:59:59 GMT+2
- **Last-Modified** : date/heure à laquelle la ressource a été modifiée pour la dernière fois
 - Ex: **Last-Modified**: Tue, 31 Mar 2015 12:12:12 GMT+2

Structure d'une réponse

1. Ligne de statut: HTTP-version Status-code Reason-phrase

- Ex: HTTP/1.1 200 OK

2. Entêtes (ordre conseillé mais pas obligatoire)

- Entêtes **généraux**: Applicables aux requêtes **et** réponses
 - Ex: Date: Thu, 26 Mar 2015 14:48:50 GMT)
- Entêtes de **réponse**: Spécifiques aux réponses
 - Ex: Server: Apache
- Entêtes d'**entité**: méta-informations concernant le corps du message
 - Ex: Content-Length: 204

3. Ligne vide

4. Corps du message (**données**)

- Ex: texte, code html, données de formulaire, image, etc.

Ligne de statut

- **Status-code** : code numérique représentant l'état de la requête (succès, échec, etc.)
 - 1xx: Informations
 - 2xx: Succès
 - 3xx: Redirection
 - 4xx: Erreur client
 - 5xx: Erreur serveur
- **Reason-phrase** : texte expliquant le **Status-code**

Les principaux codes de statut

Code	Signification	Code	Signification	Code	Signification
100	Continue	101	Switching protocols		
200	OK	201	Created	202	Accepted
203	Non-Authoritative Information	204	No Content	205	Reset Content
206	Partial Content				
300	Multiple Choices	301	Moved Permanently	302	Found
303	See Other	304	Not Modified	305	Use Proxy
306		307	Temporary Redirect	308	Permanent Redirect
400	Bad Request	401	Unauthorized	402	Payment Required
403	Forbidden	404	Not Found	405	Method Not Allowed
406	Not Acceptable	407	Proxy Authentication Required	408	Request Timeout
409	Conflict	410	Gone	411	Length Required
412	Precondition Failed	413	Request Entity Too Large	414	Request-URI Too Long
415	Unsupported Media Type	416	Requested Range Not Satisfiable	417	Expectation Failed
500	Internal Server Error	501	Not Implemented	502	Bad Gateway
503	Service Unavailable	504	Gateway Timeout	505	HTTP Version Not Supported

Réponse: les entêtes spécifiques

- **Accept-Ranges** : indique si le serveur accepte les envois partiels
 - Ex: **Accept-Ranges: bytes** ou **Accept-Ranges: none**
- **Age** : age (en secondes) des données renvoyées (utile si les données sont en cache)
 - Ex: **Age: 1030**
- **ETag** : chaine (hash) identifiant de manière unique **une version** d'une ressource.
 - Ex: **ETag: "686897696a7c876b7e"**
- **Location** : demande au client d'effectuer une redirection vers l'URL fournie
 - Ex: **Location: http://www.nouvelle-adresse.fr**
- **Proxy-Authenticate** : Demande d'authentification du client par un proxy
 - Ex: **Proxy-Authenticate: Basic realm="WallyWorld"**
- **Retry-After** : demande de re-tenter la requête après n secondes (après erreur 503: Service unavailable)
 - Ex: **Retry-After: 120**

Réponse: les entêtes spécifiques

- **Server** : information concernant le serveur qui a répondu à la requête
 - Ex: `Server: Apache/2.2.14 (Win32)`
- **Set-Cookie** : ensemble de paires clé/valeur à stocker sur le client + options
 - Ex: `Set-Cookie: name1=value1,name2=value2; Expires=Wed, 09 Jun 2021 10:18:14 GMT`
- **Vary** : informe le client qu'il existe plusieurs "versions" de la ressource demandée (cf. négociation de contenu)
 - Ex: `Vary: Accept-Language, Accept-Encoding`
- **WWW-Authenticate** : Demande d'authentification du client par un serveur web
 - Ex: `WWW-Authenticate: Basic realm="WallyWorld"`

Retour sur la négociation de contenu

- Possibilité d'avoir plusieurs versions d'une même ressource
 - On peut demander au serveur des informations sur la ressource via une requête `HEAD`. Le header `Vary` indique sur quels critères on peut négocier.
 - Le client spécifie ses préférences avec les entêtes `Accept-*` lors de sa requête `GET`
 - Si le serveur possède une version adéquate il la renverra. Sinon il renvoie un code d'erreur `406 Not Acceptable`
 - On peut "négocier": la langue, l'encodage des caractères, le type MIME, la compression
- **Astuce** pour les fichiers **image, audio et video**: si on ne précise pas d'extension dans son fichier HTML, le navigateur va négocier le format le plus approprié.

Les cookies

- HTTP est un protocole sans état
 - Chaque requête est indépendante de la précédente
 - Comment garder une trace des transactions passées (gestion de **session**, **personnalisation**, **tracking**) ?
- Les cookies HTTP !
 - Entête `Set-Cookie` envoyé par le serveur pour stocker des informations sur le client
 - Entête `Cookie` utilisé par le client pour renvoyer vers le serveur les informations stockées précédemment
 - Taille d'un cookie limitée à 4096 octets
 - Max 50 cookies par domaine (site web)
 - Les cookies ont une date d'expiration

Les cookies

- Sécurité...
 - Un serveur ne peut récupérer que les cookies qu'il a lui même demandé au client de stocker (même domaine)
- Alternatives
 - Passage d'identifiant de session dans la partie *query string* l'URL (GET)
 - Passage d'identifiant de session dans un champ de formulaire caché (POST).
 - [Local storage et session storage](#) de HTML5

Les cookie: exemple

Requête du client:

```
GET /index.html HTTP/1.1
Host: www.univ-nantes.fr
...
```

Réponse du serveur:

```
HTTP/1.0 200 OK
Content-type: text/html
Set-Cookie: theme=blue
Set-Cookie: sessionToken=fds45fds2; Expires=Fri, 26 Jun 2017 10:10:30 GMT+2
...
```

Autre requête (ultérieure) du client:

```
GET /cours.html HTTP/1.1
Host: www.univ-nantes.fr
Cookie: theme=blue; sessionToken=fds45fds2
...
```

Les webSockets

- Pourquoi ?
 - HTTP est basé sur une architecture client / serveur
 - Le serveur ne peut **que répondre** à une requête du client
 - Pas de vraies possibilités communication bi-directionnelle (ex: *Push*)
 - *Polling* et *long-polling* = requêtes périodiques de l'état d'une ressource
- C'est quoi ?
 - Protocole de communication faisant partie des spécifications HTML5
 - Complémentaire à HTTP
 - **Canal de communication bi-directionnelle** (*full-duplex*)
 - Moins verbeux
 - Latence moindre
 - Traverse simplement les proxy et firewalls

Le protocole

- Doit fonctionner avec l'infrastructure web existante
 - **Démarre par une requête HTTP**
 - Demande de **changement de protocole** via l'entête `Upgrade` et un échange de clés `Sec-WebSocket-Key` et `Sec-WebSocket-Accept`
 - Passage ensuite en websocket sur la **même connexion TCP/IP** (par défaut port 80 ou 443)
- Contrairement à HTTP, les trames WebSocket utilisent un **encodage binaire**
- Très **peu de données additionnelles liées au protocole** (entête, etc.) : quelques octets

WebSockets et JavaScript

- Côté navigateur
 - Fait partie de **HTML5**
 - **API** clairement définie et universelle
- Côté **serveur web**: dépendant de la technologies utilisée
 - Apache WebSocket module: module (écrit en C) pour apache
 - pywebsocket4 : implémentation en python, pour apache également
 - jWebSocket
 - Socket.io, **ws**: modules WebSockets pour **nodejs**
 - Etc.

Websockets coté navigateur (HTML5)

Exemple minimaliste

```
// ouverture de la connexion
var ws = new WebSocket("ws://exemple.polytech-nantes.fr");

// on déclare un "callback" qui sera appelé à l'ouverture de la connexion
ws.on('open', function() {
    // envoi d'un message de bienvenue
    ws.send('Je suis un navigateur !');
});

// autre "callback" appelé à la réception d'un message
ws.on('message', function(message) {
    console.log('Message reçu (client): %s', message);
});
```

Websockets coté serveur web (ws/nodejs)

Exemple minimaliste

```
// import du module nodejs "ws"
var WebSocketServer = require('ws').Server

// ouverture de la connexion (ici sur le port 8080)
var wss = new WebSocketServer({port: 8080});

// callback appelé à l'ouverture de la connexion
wss.on('connection', function(ws) {
  // un fois la connexion ouverte, on déclare un "callback" qui sera appelé
  // à la réception d'un message
  ws.on('message', function(message) {
    console.log('Message reçu (serveur): %s', message);
  });
  // on envoie ensuite un message au navigateur qui vient de se connecter
  ws.send('Moi, je suis un serveur...');
});
```

Fin des protocoles

Next... Programmation serveur

